

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN HIỆU**

**TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

**BÀI TẬP LỚN**

**MÔN: KỸ THUẬT LẬP TRÌNH**

Giảng viên: ThS.TRẦN PHONG NHÃ

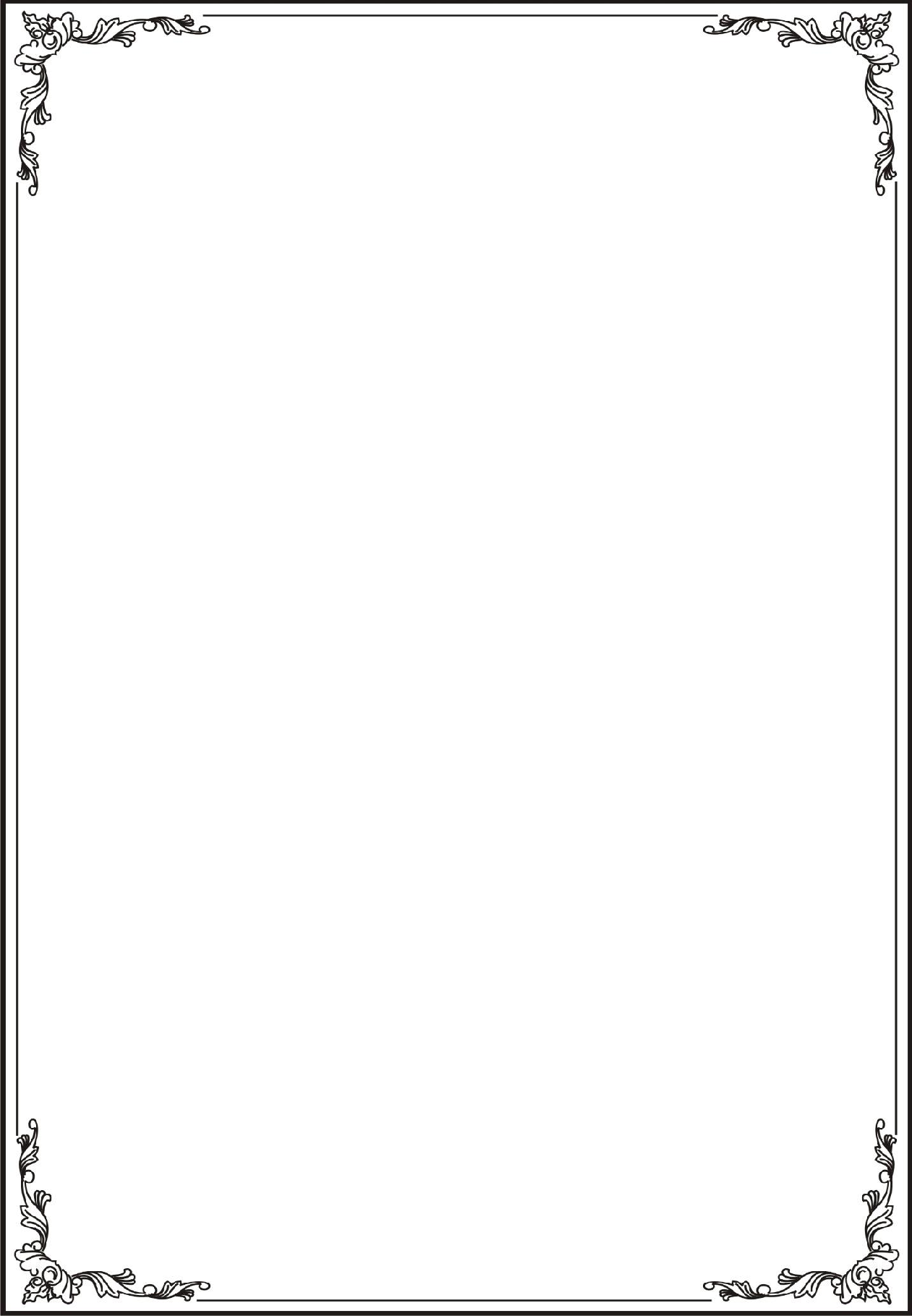
Sinh viên thực hiện: NGUYỄN KHÁNH HƯNG

Lớp: CQ.65.CNTT

Khóa: 65

MSSV: 6551071041

Tp. Hồ Chí Minh, tháng 5 năm 2025



**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN HIỆU**

**TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

**BÀI TẬP LỚN**

**MÔN: KỸ THUẬT LẬP TRÌNH**

Giảng viên: ThS.TRẦN PHONG NHÃ

Sinh viên thực hiện: NGUYỄN KHÁNH HƯNG

Lớp: CQ.65.CNTT

Khóa: 65

MSSV: 6551071041

Tp. Hồ Chí Minh, tháng 5 năm 2025

**LỜI CẢM ƠN**

Lời nói đầu tiên, em xin gửi tới Quý Thầy Cô Bộ môn Công nghệ Thông tin Trường Đại học Giao thông vận tải phân hiệu tại thành phố Hồ Chí Minh lời chúc sức khỏe và lòng biết ơn sâu sắc.

Em xin chân thành cảm ơn quý thầy cô đã giúp đỡ tạo điều kiện để em hoàn thành bài tập lớn này. Đặc biệt em xin cảm thầy Trần Phong Nhã đã nhiệt tình giúp đỡ, hướng dẫn cho em kiến thức, định hướng và kỹ năng để có thể hoàn thành bài báo cáo này này.

Tuy đã cố gắng trong quá trình nghiên cứu tìm hiểu tuy nhiên do kiến thức còn hạn chế nên vẫn còn tồn tại nhiều thiếu sót. Vì vậy em rất mong nhận được sự đóng góp ý kiến của Quý thầy cô bộ môn để đề tài của em có thể hoàn thiện hơn.

Lời sau cùng, em xin gửi lời chúc tới Quý Thầy Cô Bộ môn Công nghệ thông tin và hơn hết là v có thật nhiều sức khỏe, có nhiều thành công trong công việc. Em xin chân thành cảm ơn!

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

***Tp. Hồ Chí Minh, ngày ….… tháng ….… năm ….…***

**Giảng viên hướng dẫn**

**ThS.Trần Phong Nhã**

**MỤC LỤC**

[**LỜI CẢM ƠN** 3](#_Toc198917584)

[**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN** 4](#_Toc198917585)

[**MỤC LỤC** 5](#_Toc198917586)

[**1. Hàm** 6](#_Toc198917587)

[1.1. Các khái niệm 6](#_Toc198917588)

[1.2. Xây dựng hàm 6](#_Toc198917589)

[1.3. Sử dụng hàm 7](#_Toc198917590)

[1.4. Nguyên tắc hoạt động của hàm 7](#_Toc198917591)

[**2. Con trỏ** 9](#_Toc198917592)

[2.1. Địa chỉ 9](#_Toc198917593)

[2.2. Con trỏ 9](#_Toc198917594)

[**3. Con trỏ mảng** 11](#_Toc198917595)

[3.1. Phép toán lấy địa chỉ áp dụng được cho các phần tử của mảng một chiều 11](#_Toc198917596)

[3.2. Tên mảng là một hằng địa chỉ 11](#_Toc198917597)

[3.3. Nếu con trỏ pa trỏ tới một phần tử a[k] nào đó thì: 11](#_Toc198917598)

[**4. Mảng con trỏ** 14](#_Toc198917599)

[**5. Con trỏ hàm** 15](#_Toc198917600)

[5.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm 15](#_Toc198917601)

[5.2. Tác dụng của con trỏ hàm 15](#_Toc198917602)

[5.3. Đối con trỏ hàm 16](#_Toc198917603)

[**6. Cấp phát động** 17](#_Toc198917604)

[**7. Xử lý tệp** 18](#_Toc198917605)

[7.1. Các hàm dùng chung cho cả hai loại tệp 18](#_Toc198917606)

[7.2. Các hàm chỉ dùng cho tệp nhị phân 19](#_Toc198917607)

[7.3. Các hàm chỉ dùng cho tệp văn bản 19](#_Toc198917608)

[7.4. Nhập xuất ngẫu nhiên 19](#_Toc198917609)

[7.4.1. Hàm rewind 19](#_Toc198917610)

[7.4.2. Hàm fseek 19](#_Toc198917611)

[7.4.3. Hàm ftell 20](#_Toc198917612)

[7.4.4.Ví dụ: 20](#_Toc198917613)

[**8. Kiểu cấu trúc** 21](#_Toc198917614)

[**9.Danh sách liên kết** 24](#_Toc198917615)

**1. Hàm**

1.1. Các khái niệm

Liên quan đến hàm có các khái niệm sau:

* Tên hàm
* Kiểu giá trị của hàm
* Đối hay tham số hình thức
* Thân hàm
* Khai báo hàm ( nguyên mẫu – prototype )
* Lời gọi hàm
* Tham số thực

Đối với mỗi hàm bao giờ cũng có 2 giai đoạn khác nhau là: xây dựng hàm và sử dụng hàm.

1.2. Xây dựng hàm

Xây dựng hàm bao gồm các việc: Khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra các câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau:

Type ten\_ham (khai báo các đối)  
{  
khai báo các biến cục bộ  
các câu lệnh  
[return [bieu thuc];]  
}

Chú ý 1: Đối với các hàm không cho giá trị (như thủ tục trong Pascal) thì dùng kiểu void. Ví du hàm dùng để hiển thị giá trị lớn nhất của ba đối thục có thể viết như sau:

void ht\_max\_3s(float a,float b,float c)  
{  
float x;  
x = a>b?a:b;  
printf("\n Max= %0.2f",x>c?x:c);  
}  
Chú ý 2: Hàm không đối thì ta nên dùng void để khai báo đối, ví dụ hàm báo thức có thể viết như sau:  
void bao\_thuc(void)  
{  
int i;  
for(i=0; i<10; ++i) putch(7);  
}

Chú ý 3: Khi gặp một toán tử return có chứa biểu thức, thì giá trị của biểu thức sẽ được chuyển kiểu cho phù hợp với kiểu của hàm hàm trước khi nó được gán cho hàm.

1.3. Sử dụng hàm

Hàm được sử dụng thông qua lời gọi tới nó. Cách viết một lời gọi hàm như sau:  
ten\_ham([danh sách các tham số thực])  
Ở đây ta cần lưu ý:  
+ Số tham số thực phải bằng số đối.  
+ Kiểu của tham số thực phải phù hợp với kiểu của đối tương ứng.

1.4. Nguyên tắc hoạt động của hàm

Một cách tổng quát lời gọi hàm có dạng sau:

tên\_hàm ([ danh sách tham số thực ])

Một điều cần nhớ khi viết lời gọi hàm là: số tham số thực phải bằng số tham

số hình thức ( đối ) và mỗi tham số thực phải có cùng kiểu giá trị như kiểu giá trị

của đối tương ứng với nó.

Khi gặp một lời gọi hàm thì hàm bắt đầu được thực hiện. Nói cách khác, khi

máy gặp một lời gọi hàm ở một chỗ nào đó của chương trình, thì máy sẽ tạm rời

chỗ đó và chuyển đến hàm tương ứng. Qúa trình đó sẽ diễn ra theo trình tự 4

bước như sau:

a/ Cấp phát bộ nhớ cho các đối và các biến cục bộ.

b/ Gán giá trị của các tham số thực cho các đối tương ứng.

c/ Thực hiện các câu lệnh trong thân hàm.

d/ Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối, các biến cục bộ (giải phóng bộ nhớ của các đối, biến cục bộ) và thoát khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

**2. Con trỏ**

2.1. Địa chỉ

Liên quan đến một biến ta đã có khái niệm:

- Tên biến  
- Kiểu biến  
- Giá trị của biến

Ví dụ câu lệnh  
float alpha = 30.5;

Xác định một biến có tên là alpha có kiểu float và có giá trị 30.5. Ta cũng đã biết, theo khai báo trên, máy sẽ cấp phát cho biến alpha một khoảng nhớ gồm 4 byte liên tiếp. Địa chỉ của biến là số thứ tự của byte đầu tiên trong môt dẫy các byte liên tiếp mà máy dành cho biến (các byte được đánh số từ 0).  
Một điều cần chú ý là mặc dù địa chỉ của biến là một số nguyên nhưng không được đánh đồng nó với các số nguyên thông thường dùng trong các phép tính.  
Rõ ràng địa chỉ của hai biến kiểu int liên tiếp cách nhau 2 byte, địa chỉ của hai biến kiểu float liên tiếp cách nhau 4 byte.... Nên máy sẽ phân biệt các kiểu địa chỉ: Địa chỉ kiểu int, kiểu float, kiểu double,... . Phép toán &x cho ta địa chỉ của biến x.

2.2. Con trỏ

Con trỏ là một biến dùng để chứa địa chỉ. Vì có nhiều loại địa chỉ nên cũng có nhiều kiểu con trỏ tương ứng. Con trỏ kiểu int dùng để chứa địa chỉ các biến kiểu int. Tương tự, ta có con trỏ kiểu float, kiểu double.... Cũng như đối với bất kỳ một biến nào khác, một con trỏ cần khai báo trước khi sử dụng. Việc khai báo biến con trỏ được thực hiện theo mẫu sau:

type \*tên\_con\_trỏ;  
Ví dụ câu lệnh  
int x,y,\*px,\*c;  
khai báo hai biến kiểu int là x,y và hai con trỏ kiểu int là px và c.  
Tương tự câu lệnh  
float \*t, \*d;  
khai báo hai con trỏ kiểu float là t và d.  
Khi đã có các khai báo trên thì các câu lệnh  
c= &y;  
px = &x;

hoàn toàn xác định. Câu lệnh thứ nhất sẽ gán địa chỉ của y cho con trỏ c và câu lệnh thứ hai sẽ gán địa chỉ của biến x cho con trỏ px. Như vậy trong con trỏ c chứa địa chỉ của biến y và trong con trỏ px chứa địa chỉ của biến x. Chú ý rằng nếu viết

t = &y;

thì sẽ nhận được một câu lệnh sai: t là con trỏ kiểu float, nó chỉ chứa được địa chỉ của các biến float. Câu lệnh trên nhằm gán địa chỉ của biến nguyên y cho con trỏ

t là không thể chấp nhận được.  
  
Chú ý: Khi con trỏ px chứa địa chỉ của biến x thì ta nói px trỏ tới x.

**3. Con trỏ mảng**

Trong C có mối quan hệ chặt chẽ giữa con trỏ và mảng: Các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ. Trong mục này sẽ trình bầy các vấn đề có liên quan đến mảng và con trỏ.

3.1. Phép toán lấy địa chỉ áp dụng được cho các phần tử của mảng một chiều

Giả sử ta có khai báo  
double b[20];  
khi đó phép toán  
&b[i]  
với i trong khoảng [0,19] cho địa chỉ của phần tử b[i].

3.2. Tên mảng là một hằng địa chỉ

Như đã biết với khai báo  
float a[10];

máy sẽ bố trí cho mảng a mười khoảng nhớ liên tiếp (mỗi khoảng nhớ 4 byte). Như vậy nếu biết địa chỉ của một phần tử nào đó của mảng a, thì dễ dàng suy ra địa chỉ của các phần tử khác. Một sự kiện mà ta định nói ở đây là: Tên mảng là một hằng địa chỉ, nó chính là địa chỉ của phần tử đầu tiên của mảng. Như vậy, trong mọi ngữ cảnh:

a tương đương với &a[0]  
a+i tương đương với &a[i]  
\*(a+i) tương đương với a[i]

3.3. Nếu con trỏ pa trỏ tới một phần tử a[k] nào đó thì:

pa+i trỏ tới phần tử thứ i sau a[k], tức là a[k+i]  
pa-i trỏ tới phần tử thứ i trước a[k], tức là a[k-i]  
\*(pa+i) tương đương với pa[i]  
Như vậy sau hai câu lệnh  
float a[30], \*p;  
p = a;  
thì bốn cách viết sau có tác dụng như nhau:  
a[i] \*(a+i) \*(p+i) p[i]  
Bây giờ ta đưa ra một ví dụ minh họa các điều nêu trên.

Xét một bài toán đơn giản: Vào từ bàn phím giá trị các phần tử của một mảng và tính tổng của chúng. Dưới đây là bốn chương trình giải bài toán nêu trên.

/\* vao so lieu cho mang va tinh tong, ban 1 \*/  
#include "stdio.h"  
main()  
{  
float a[4],s;  
int i;  
for (i=0;i<4; ++i )  
{  
printf("\n a[%d] = ",i);  
scanf("%f",&a[i]);  
}  
s = 0;  
for (i=0; i<4; ++i )  
s += a[i];  
printf("\n tong = %8.2f",s);  
}

/\* vao so lieu cho mang va tinh tong , ban 2 \*/  
#include "stdio.h"  
main()  
{  
float a[4],s;  
int i;  
for (i=0; i<4; ++i )  
{  
printf("\n a[%d] = ",i);  
scanf("%f", a+i );  
}  
s = 0;  
for(i=0; i<4; ++i )  
s += a[i];  
printf("\n tong = %8.2f",s);  
}

/\* vao so lieu cho mang va tinh tong , ban 3 \*/  
#include "stdio.h"  
main()  
{  
float a[4],s,\*pa;  
int i;  
pa = a;  
for (i = 0; i<4; ++i )  
{  
printf("\n a[%d] = ",i);  
scanf("%f",&pa[i] );  
}  
s = 0;  
for(i=0; i<4; ++i )  
s += pa[i];  
printf("\n tong = %8.2f",s);  
}

# **4. Mảng con trỏ**

Mảng con trỏ là sự mở rộng khái niệm con trỏ. Mảng con trỏ là một mảng mà mỗi phần tử của nó có thể chứa được một địa chỉ nào đó. Cũng giống như con trỏ, mảng con trỏ có nhiều kiểu: Mỗi phần tử của mảng con trỏ kiểu int sẽ chứa được địa chỉ kiểu int. Tương tự, ta suy ra ý nghĩa của các mảng con trỏ kiểu char , float,

... Mảng con trỏ được khai báo theo mẫu

type \*namearr[N];

trong đó type có thể là int, float, double, char, ..., name arr là tên của mảng, N là

một hằng số nguyên xác định độ lớn của mảng.

Khi gặp khai báo trên máy sẽ cấp phát N khoảng nhớ liên tiếp cho N phần tử của

mảng namearr. Ví dụ, câu lệnh

double \*pa[100];

khai báo một mảng con trỏ kiểu double gồm một trăm phần tử. Mỗi phần tử pa[i] có thể dùng để lưu trữ một địa chỉ kiểu double. Chú ý rằng bản thân mảng con trỏ không thể dùng để lưu trữ số liệu. Tuy nhiên mảng con trỏ cho phép sử dụng các mảng khác để lưu trữ số liệu một cách có hiệu qủa hơn theo cách: Chia mảng thành các phần và ghi nhớ địa chỉ đầu của mỗi phần vào một phần tử của mảng

con trỏ

Cũng nên lưu ý rằng trước khi sử dụng một mảng con trỏ cần gán cho mỗi phần tử của nó một giá trị. Giá trị này phải là địa chỉ của một biến hoặc của một phần tử mảng. Các phần tử của mảng con trỏ kiểu char có thể được khởi đầu bằng các

xâu ký tự.

# **5. Con trỏ hàm**

## 5.1. Cách khai báo con trỏ hàm và mảng con trỏ hàm

Ta trình bầy quy tắc khai báo thông qua các ví dụ  
+ Tác dụng của câu lệnh:  
float (\*f)(float), (\*mf[50])(int);  
là khai báo:  
- f là con trỏ hàm kiểu float có đối float,  
- mf là mảng con trỏ hàm kiểu float có đối int (mảng có 50 phần tử).  
+ Tác dụng câu lệnh:  
double (\*g)(int, double), (\*mg[30])(double, float);  
là khai báo:  
- g là con trỏ hàm kiểu double có các đối int và double,  
- mg là mảng con trỏ hàm kiểu double có các đối double và float (mảng có 30 phần tử).

## 5.2. Tác dụng của con trỏ hàm

Con trỏ hàm dùng để chứa địa chỉ của hàm. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm. Các ví dụ sau minh hoạ điều này.

Ví dụ 1: Vừa khai báo vừa gán.  
#include <stdio.h>  
double fmax(double x, double y) /\* Hàm tính max \*/  
{  
return (x>y?x:y);  
}  
/\* Khai báo và gán tên hàm cho con trỏ hàm \*/  
double (\*pf)(double, double) = fmax;  
/\* Sử dụng con trỏ hàm \*/  
main()  
{  
printf("\nmax= %f",pf(4.5,78.9));  
}

Ví dụ 2: Gán sau khi khai báo.  
#include <stdio.h>  
double fmax(double x, double y) /\* Hàm tính max \*/  
{  
return (x>y?x:y);  
}  
/\* Khai báo con trỏ hàm \*/  
double (\*pf)(double, double);  
main()  
{  
/\* Gán tên hàm cho con trỏ hàm \*/  
pf = fmax;  
/\* Sử dụng con trỏ hàm \*/  
printf("\nmax= %f",pf(4.5,78.9));  
}

## 5.3. Đối con trỏ hàm

C cho phép thiết kế các hàm mà tham số thực trong lời gọi tới nó lại là tên của một hàm khác. Khi đó tham số hình thức tương ứng phải là một con trỏ hàm.  
Cách dùng con trỏ hàm trong thân hàm. Điều này có thể diễn đạt như sau.

Nếu đối được khai báo:  
double (\*f)(double, int)  
thì trong thân hàm ta có thể dùng các cách viết sau để xác định giá trị của hàm (do con trỏ f trỏ tới):  
f(x,m) (f)(x,m) (\*f)(x,m)  
ở đây x là biến double, m là biến int.

Ví dụ : Chương trình dưới đây sẽ:  
- Lập hàm tính tích phân của f(x) trên đoạn [a,b] theo phương pháp hình thang bằng cách chia [a,b] thành 1000 khoảng có độ dài như nhau.  
- Dùng hàm trên để tính:  
S1 = Tích phân trên [0,PI/2] của sin(x)  
S2 = Tích phân trên [0,PI/2] của cos(x)  
S3 = Tích phân trên [0,1.0] của exp(x)  
S4 = Tích phân trên [-1.2,3.5] của  
g(x) = (exp(x)-2\*sin(x\*x))/(1+pow(x,4));  
/\* Chương trình tính tích phân \*/  
#include "stdio.h"  
#include "math.h"  
double tp(double (\*f)(double), double a, double b);  
double g(double);

# **6. Cấp phát động**

-Cấp phát động bộ nhớ

Biến dùng để lưu trữ dữ liệu trong chương trình. Biến cần được khai báo trước khi sử dụng. Thực chất của việc khai báo biến là xin cấp phát một vùng bộ nhớ để lưu trữ dữ liệu. Việc khai báo biến đã đề cập trong các chương trước chính là việc cấp phát tĩnh bộ nhớ. Tuy nhiên, không phải khi nào số lượng và kích thước của các biến cũng xác định được ngay khi biên dịch, chẳng hạn nó có thể phụ thuộc vào các thông tin người dùng cung cấp khi thực hiện chương trình. Trong trường hợp này, bộ nhớ cần được cấp phát động.

Để cấp phát động bộ nhớ, sử dụng hàm malloc trong thư viện stdlib.h:  
void \*malloc(size\_t size)  
Trong đó:  
size là kích thước vùng nhớ cần cấp.  
size\_t là một kiểu dữ liệu định sẵn trong thư viện stdlib.h (thông thường có thể khai báo kiểu unsigned int).  
Hàm trả về con trỏ kiểu void chỉ đến ô nhớ đầu tiên của vùng nhớ được cấp. Nếu bộ nhớ không còn đủ, giá trị con trỏ là NULL.  
Để giải phóng vùng nhớ được trỏ bởi con trỏ p, sử dụng hàm free trong thư viện stdlib.h:  
void free(void \*p)  
Lưu ý, nếu giá trị p bằng NULL thì hàm free sẽ không làm gì.  
Ví dụ: Nhập một dãy các số nguyên và tính tổng của dãy đó.

Nếu giải quyết bài toán theo cách đã biết (cấp phát tĩnh), ta cần phải khai báo một mảng với kích thước tối đa mà ta dự kiến. Điều này có thể gây lãng phí bộ nhớ. Sử dụng cấp phát động bộ nhớ thì dãy có bao nhiêu số, ta chỉ cấp bộ nhớ để lưu

trữ đúng bấy nhiêu.  
Dưới đây là chương trình.  
#include "stdio.h"  
#include "stdlib.h"  
int main()  
{  
int \*p, i, n, s= 0;  
printf("So phan tu= ");  
scanf("%d",&n);  
p= (int\*)malloc(n\*sizeof(int));  
for(i= 0; i< n; ++i){  
printf("So thu %d= ", i);  
scanf("%d", (p+i));  
s= s+ \*(p+i);  
}  
printf("Tong cua day so= %d",s);  
free(p);  
}

# **7. Xử lý tệp**

Các thao tác trên tệp tin

## 7.1. Các hàm dùng chung cho cả hai loại tệp

Hàm fopen  
+ Dạng hàm:  
FILE \*fopen(const char \*tên\_tệp, const char \*kiểu);  
+ Công dụng: Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp 2 sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm trả về giá trị NULL.

+ Các đối:  
Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.  
Kiểu có thể có các giá trị sau:  
Chú ý: Trong các kiểu đọc/ghi, cần làm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc từ ghi sang đọc. Các hàm fflush và hàm di chuyển đầu từ đều làm được chuyện này.  
Hàm fclose  
+ Dạng hàm:  
int fclose(FILE \*fp);

+ Công dụng: Hàm dùng để đóng tệp. Nội dung đóng tệp gồm: Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi), xoá vùng đệm (khi đang đọc) và giải phóng biến fp để nó có thể dùng cho tệp khác. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

+ Đối: fp là con trỏ tương ứng với tệp cần đóng.  
Hàm fcloseall  
+ Dạng hàm:  
int fcloseall(void);  
+ Công dụng: Hàm dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đóng được, trái lại hàm cho EOF.  
Hàm fflush  
+ Dạng hàm:  
int fflush(FILE \*fp);  
+ Công dụng: Hàm dùng làm sạch vùng đệm của tệp fp. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.  
+ Đối: fp là con trỏ tệp.  
Hàm fflushall  
+ Dạng hàm:  
int fflushall(void);  
+ Công dụng: Hàm dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đang mở, trái lại hàm cho EOF.  
Hàm feof  
+ Dạng hàm:  
int feof(FILE \*fp);  
+ Công dụng: Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.  
+ Đối: fp là con trỏ tệp.

## 7.2. Các hàm chỉ dùng cho tệp nhị phân

putw: dùng để ghi một số nguyên (2 byte) lên tệp.  
getw: dùng để đọc một số nguyên (2 byte) từ tệp.  
fwrite: dùng để ghi một số mẫu tin lên tệp.  
fread: dùng để đọc một số mẫu tin từ tệp.

## 7.3. Các hàm chỉ dùng cho tệp văn bản

fprintf: dùng để ghi dữ liệu theo khuôn dạng lên tệp.  
fscanf: dùng để đọc dữ liệu từ tệp theo khuôn dạng.  
fputs: dùng để ghi một chuỗi ký tự lên têp.  
fgets: dùng để đọc một dẫy ký tự từ tệp.

## 7.4. Nhập xuất ngẫu nhiên

Mỗi tệp khi đang mở có một con trỏ chỉ vị dùng để xác định vị trí đọc ghi trên

tệp. Khi mở tệp tin để đọc hay ghi, con trỏ chỉ vị luôn luôn ở đầu tệp tin (byte

0). Nhưng nếu tệp được mở theo mode “” thì con trỏ chỉ vị ở cuối tệp để ghi thêm dữ liệu vào tệp này. Việc nhập xuất dữ liệu được bắt đầu từ vị trí hiện tại của con trỏ chỉ vị và sau khi hoàn thành thì con trỏ này dịch chuyển một số byte chính bằng số byte đã đọc hay ghi. Như vậy việc nhập xuất được tiến

hành tuần tự theo chiều từ đầu đến cuối tệp tin.

Trong mục này sẽ nói cách di chuyển con trỏ chỉ vị đến vị trí mong muốn, nhờ

đó ta có thể nhập xuất tại bất kỳ chỗ nào trên tệp tin. Cách làm này gọi là nhập

xuất ngẫu nhiên rất tiện lợi cho việc sữa chữa, bổ sung dữ liêu trực tiếp trên tệp.

### 7.4.1. Hàm rewind

+ Dạng hàm:

void rewind(FILE \*fp);

+ Công dụng: Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập

xuất trên tệp fp được thực hiện từ đầu tệp.

+ Đối:

fp là con trỏ tệp.

### 7.4.2. Hàm fseek

+ Dạng hàm:

int fseek(FILE \*fp, long sb, int xp);

+ Công dụng: Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp

qua một số byte bằng giá trị tuyệt đối cua sb. Chiều di chuyển là về cuối tệp nếu

sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

+ Đối:

fp là con trỏ tệp,

sb là số byte cần di chuyển,

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đấy

xp có thể nhận các giá trị sau:

xp = SEEK\_SET hay 0: Xuất phát từ đầu tệp.

xp = SEEK\_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp = SEEK\_END hay 2: Xuất phát từ cuối tệp.

Chú ý: Không nên dùng fseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ

làm cho việc định vị thiếu chính xác.

### 7.4.3. Hàm ftell

+ Dạng hàm:

long ftell(FILE \*fp);

+ Công dụng: Khi thành công hàm cho biết vị trí hiện tại của con trỏ chỉ vị

(byte thứ mấy trên tệp fp). Số thứ tự của byte được tính từ 0. Khi có lỗi hàm trả

về -1L.

+ Đối:

fp là con trỏ tệp.

### 7.4.4.Ví dụ:

Chương trình dưới đây dùng các hàm fseek và ftell để xác định độ dài

của tệp bất kỳ.

/\* Dung fseek va ftell xac dinh do dai tep tin \*/

#include <stdio.h>

#include <conio.h>

main()

{

FILE \*fp;

long n;

char tep[14];

clrscr();

printf(“\nTen tep”); gets(tep);

fp=fopen(tep”rb”);

if(fp==NULL)

{

Printf”Khong ton tai %s”,tep);

exit(1);

}

fseek(fp,0,SEEK\_END);

n=ftell(fp);

fclose(fp);

printf”\nDo dai cua tep %s la %ld byte”tep,n);

}

# **8. Kiểu cấu trúc**

Để lưu trữ và xử lý thông tin trong máy tính ta có các biến và các mảng. Mỗi biến chứa được một giá trị. Mảng có thể xem là tập hợp nhiều biến có cùng một kiểu giá trị và được biểu thị bằng một tên. Cấu trúc có thể xem như một sự mở rộng của các khái niệm biến và mảng, nó cho phép lưu trữ và xử lý các dạng thông tin phức tạp hơn. Cấu trúc là một tập hợp các biến, các mảng và được biểu thị bởi

một tên duy nhất.  
  
Để định nghĩa một cấu trúc, sử dụng từ khóa struct theo mẫu sau:  
struct tên\_cấu\_trúc  
{  
Khai báo các thành phần  
};

Lưu ý, thành phần của một cấu trúc có thể là một biến kiểu cơ bản hoặc có thể là một cấu trúc khác.

Ví dụ:  
struct ngay  
{  
int ngay\_thu;  
char ten\_thang[10];  
int nam;  
};  
  
struct sinh\_vien  
{  
char ten[20];  
struct ngay ngay\_sinh;  
char dia\_chi[20];  
};  
  
Để định nghĩa một kiểu cấu trúc, sử dụng từ khóa typedef theo mẫu sau:  
typedef struct  
{  
Khai báo các thành phần  
} kiểu\_cấu\_trúc;  
  
Sau khi xây dựng được kiểu\_cấu\_trúc, nó có thể được dùng để khai báo các biến cấu trúc.

Ví dụ:  
typedef struct  
{  
int ngay\_thu;  
char ten\_thang[10];  
int nam;  
} kieu\_ngay;  
  
struct sinh\_vien  
{  
char ten[20];  
kieu\_ngay ngay\_sinh;  
char dia\_chi[20];  
};

Các thành phần cơ bản của một cấu trúc là biến và mảng, nên một lẽ tự nhiên và cũng là một quy tắc cần ghi nhớ là việc xử lý một cấu trúc bao giờ cũng phải được

thực hiện thông qua các thành phần cơ bản của nó.

Để truy nhập tới các thành phần của cấu trúc cần thông qua tên của biến cấu trúc theo mẫu sau:  
biến\_cấu\_trúc.tên\_thành\_phần  
  
Ví dụ: Nhập họ tên và điểm của sinh viên trong một lớp, in ra danh sách các sinh viên đạt (có điểm không nhỏ hơn 5). Dưới đây là chương trình.  
#include "stdio.h"  
#include "stdlib.h"  
typedef struct  
{  
char ht[20];  
float diem;  
} kieu\_sv;  
int main()  
{  
int i, n;  
float t;  
kieu\_sv \*sv;  
printf("So sinh vien= ");  
scanf("%d",&n);  
sv= (kieu\_sv\*)malloc(n\*sizeof(kieu\_sv));  
for(i= 0; i< n; ++i)  
{  
fflush(stdin);  
printf("SV thu %d= \n", i+1);  
printf("Ho ten: ");  
gets(sv[i].ht);  
printf("Diem: ");  
scanf("%f", &t);  
sv[i].diem= t;  
}  
puts("\nDanh sach sinh vien dat\n");  
for(i=0; i<n; ++i)  
{  
if(sv[i].diem>=5)  
printf("%s%6.2f\n",sv[i].ht, sv[i].diem);  
}  
free(sv);  
}

# **9.Danh sách liên kết**

Cấu trúc có ít nhất một thành phần là con trỏ kiểu cấu trúc đang định nghĩa gọi là cấu trúc tự trỏ. Cấu trúc tự trỏ có thể định nghĩa theo một trong các cách sau đây.

Cách 1:  
typedef struct sv  
{  
char ht[25]; /\* ho ten \*/  
int diem;  
struct sv \*tiep;  
} kieu\_sv;  
  
Cách 2:  
typedef struct sv kieu\_sv;  
struct sv  
{  
char ht[25];  
int diem;  
kieu\_sv \*tiep;  
};  
  
Cách 3:  
struct sv  
{  
char ht[25];  
int diem;  
struct sv \*tiep;  
};

typedef sv kieu\_sv;  
  
Cấu trúc tự trỏ được dùng để xây dựng danh sách liên kết (móc nối), đó là một

nhóm các cấu trúc có tính chất sau:

+ Biết địa chỉ cấu trúc đầu đang được lưu trữ trong một con trỏ nào đó (giả sử pdau).

+ Trong mỗi cấu trúc (trừ cấu trúc cuối) chứa địa chỉ của cấu trúc tiếp theo của danh sách.  
+ Cấu trúc cuối chứa hằng NULL.  
  
Danh sách có 3 tính chất trên gọi là danh sách móc nối theo chiều thuận. Với danh sách này, ta có thể lần lượt truy nhập từ cấu trúc đầu tới cấu trúc cuối theo

chiếu từ trên xuống dưới.  
  
Tương tự, danh sách liên kết theo chiều ngược cũng có 3 tính chất trên nhưng theo chiều ngược lai:  
+ Biết địa chỉ cấu trúc cuối.  
+ Trong mỗi cấu trúc (trừ cấu trúc đầu) chứa địa chỉ của cấu trúc trước.  
+ Cấu trúc đầu chứa hằng NULL.

Với danh sách này, ta có thể lần lượt truy nhập từ cấu trúc cuối tới cấu trúc đầu

theo chiều từ dưới lên trên.  
Ngoài ra có thể xây dựng các danh sách mà mỗi phần tử chứa hai địa chỉ: địa chỉ

cấu trúc trước và địa chỉ cấu trúc sau. Với loại danh sách này, ta có thể truy nhập

từ trên xuống dưới theo chiều thuận hoặc từ dưới lên trên theo chiều ngược.  
Dưới đây là chương trình minh họa cách dùng danh sách móc nối.  
  
Ví dụ:  
Nhập một số sinh viên và chứa vào bộ nhớ dưới dạng danh sách móc nối thuận (lập danh sách mới), thực hiện các thao tác in danh sách ra màn hình và bổ sung một sinh viên vào cuối danh sách.  
#include "stdio.h"  
#include "stdlib.h"  
#include "conio.h"  
#include "string.h"  
typedef struct sv  
{  
char ht[25];  
float diem;  
struct sv \*tiep;  
} ksv;  
  
main()  
{  
float t;  
char ht[25];  
ksv \*pdau,\*p,\*p1;  
pdau=NULL;

/\* Nhap du lieu \*/  
while(1)  
{  
fflush(stdin);  
printf("\nHo ten(enter ket thuc):");  
gets(ht);  
if (ht[0]==0) break;  
if (pdau==NULL)  
{  
pdau=(ksv\*)malloc(sizeof(ksv));  
p=pdau;  
}  
else  
{  
p->tiep=(ksv\*)malloc(sizeof(ksv));  
p=p->tiep;  
}  
strcpy(p->ht,ht);  
printf("\nDiem: ");  
scanf("%f",&t);  
p->diem= t;  
p->tiep = NULL;  
}

/\* In danh sach lien ket ra man hinh \*/  
p=pdau;  
while(p!=NULL)  
{  
printf("\nHo ten:%s", (\*p).ht);  
printf("\nDiem:%6.2f", (\*p).diem);  
p = p->tiep;  
}

/\* Bo sung vao cuoi danh sach \*/  
printf("\nBo sung vao cuoi ");  
if(pdau==NULL)/\* Danh sach rong \*/

{

pdau=(sv\*)malloc(sizeof(ksv));

p=pdau;

}

else

{

p=pdau;

while(p->tiep!=NULL)

p=p->;tiep;

/\* cap phat vung nho, nhap \*/

p->tiep=(ksv\*)malloc(sizeof(ksv));

p=p->tiep;

}

/\* Nhap bo sung \*/

Printf(“\nHo ten: “);

gets(p->ht);

printf(“\nDiem: ”);

scanf(“%f”,&t);

p->diem=t;

p->tiep=NULL;

/\* In danh sach lien ket ra man hinh \*/

p=pdau;

while(p!=NULL)

{

printf(“\nHo ten:%s”,”,(\*p).ht);

printf(“\nDiem:%6.2f”, “,(\*p).diem);

p = p->tiep;

}

}